การเขียนโปรแกรมในภาษาระดับสูง บางครั้งเราจำเป็นต้องเรียกโปรแกรมย่อยภาษาเครื่องมาใช้ อาจจะเป็นเพราะไม่มีคำสั่งที่เราต้องการในภาษาระดับสูงนั้น หรืออาจเป็นเพราะ ถ้าเขียนด้วยภาษาระดับสูงแล้ว เครื่องจะใช้เวลาในการรันมาก ในภาษาเบสิกคุณอาจเรียกโปรแกรมย่อยภาษาเครื่องมาใช้อย่างง่ายดาย ในภาษาปาสคาลก็ได้เช่นกันโดยเฉพาะเทอร์โบปาสคาล

เทอร์โบปาสคาล กับภาษาเครื่อง

บุญเลิศ เอี่ยมทัศนา

แต่ต้องอาศัยความรู้ในเรื่องระบบจัดการกับโปรแกรมแอสเซมเบลอร์เข้าช่วย และนี่คือเหตุผลที่ทำไมผมไม่ได้เขียนเรื่องนี้เสียแต่เนิ่นๆ เพราะผมไม่เคยศึกษาระบบจัดการ CP/M-80 และไม่เคยใช้แอสเซมเบลอร์ของ Z-80 ไม่ว่าตัวใด หากคุณคิดว่าตอนนี้ผมคงศึกษาแล้ว คำตอบคือยังครับ แต่กระโดดข้ามไปศึกษาระบบจัดการ MS-DOS ที่ใช้อยู่บนเครื่องพีซีทั้งหลาย และโปรแกรม MASM ของไมโครซอฟท์ ซึ่งเป็นแอสเซมเบลอร์ของ 8086 เรื่องราวของผมตอนนี้จึงเป็นเรื่องของเทอร์โบปาสคาลเวอร์ชั่น 3.0 บนเครื่องไอบีเอ็มพีซีทั้งหลาย แต่ตอนท้ายผมจะสรุปว่า หากจะใช้กับเครื่องแอปเปิ้ล ซึ่งเป็นเทอร์โบปาสคาลเวอร์ชั่น 2.0 จะทำได้อย่างไร

เรียกใช้ภาษาเครื่อง

การใช้ภาษาเครื่องของเทอร์โบปาสคาลบนเครื่องไอบีเอ็มพีซี โดยเฉพาะพีซี/เอ็กซ์ที ทั้งแท้และเทียม ซึ่งเป็นที่แพร่หลายในบ้านเรา และต่อไปผมจะเรียกว่าไอบีเอ็ม ทำได้ 3 แบบคือ

- เรียกใช้โปรแกรมย่อยภาษาเครื่องโดยตรง(External routine)
- ทำเป็นถ้อยแถลงภาษาเครื่อง(In-line machine code)
- ผ่านการอินเตอร์รัพท์(Intr procedure)

ซึ่งผมจะกล่าวในรายละเอียดต่อไป แต่ก่อนอื่นเราจำเป็นต้องทราบในเรื่องบางเรื่องเสียก่อน

กรรมวิธีในการผ่านค่า

เรื่องแรกคือ กรรมวิธีในการผ่านค่าของภาษาปาสคาลเอง เช่น เมื่อเราใช้คำสั่ง Goto XY (40,20) คอมพายเลอร์จะดำเนินการอย่างไร เราก็ต้องไปเริ่มต้นดูที่ส่วนหัวของ
ซิเดียว GotoXY ซึ่งจะเขียนไว้เป็น

```
PROCEDURE GotoXY (x,y:Integer
```

แสดงว่าผู้เขียนโพรซิเดียว GotoXY
กำหนดไว้ว่า หากเราจะเรียกใช้โพรซิ
เราต้องผ่านค่าเป็นเลขจำนวนเต็ม 2 ตั
โดยจำนวนแรกจะเป็นค่าคอลัมน์ ส่วนจ
หลังจะเป็นค่าโรว์ ดังที่เรานำมาใช้ใน
หรือจะให้เป็นตัวแปรก็ได้

เมื่อคอมพายเลอร์อ่านถึงคำสั่ง
ขั้นตอนของการคอมพายล์ จะตรวจส
เราผ่านค่านี้ถูกต้องหรือไม่ ถ้าถูกต้อง
แกรมของเราก็ผ่านการคอมพายล์ เ
แปรคำสั่งนี้ไว้เป็นว่า

- เก็บค่า 40 ลงสแต็ค
- เก็บค่า 20 ลงสแต็ค
- CALL GotoXY

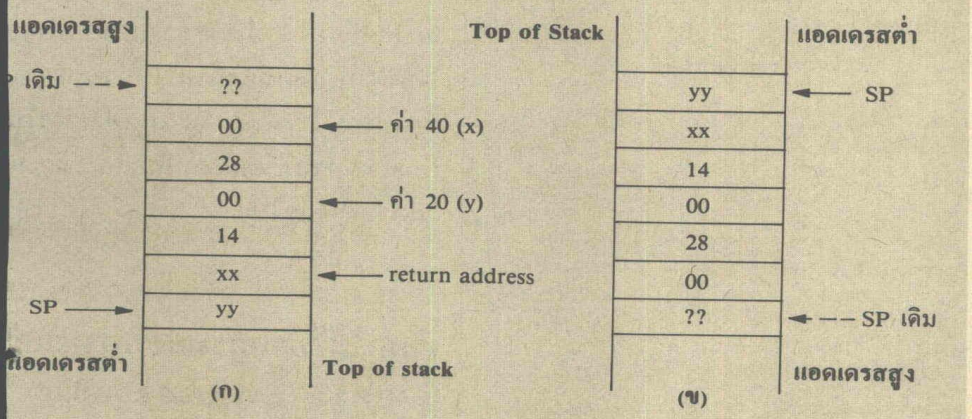
เมื่อรันจะได้ค่าในสแต็คดังแสดงใน
ที่ 1 เนื่องจากทั้ง 40 และ 20 เป็นข้อมูล
บ Integer ขนาดของข้อมูลจึงเป็น 2 ไบท์
คือ 40 จะมีค่าเป็น 0028H และ 20 จะมี
ป็น 0014H ส่วนค่า return address
นาดเป็น 2 ไบท์ เพราะเป็นค่า IP (instruc-
pointer) เท่านั้น เป็นการเรียกโปรแกรม
แบบ near นั่นคือตัวโปรแกรมย่อย Goto
จะต้องอยู่ใน code segment เดียวกันกับ
แกรมที่เรียก และนี่คือเหตุผลที่ทำให้
แกรมที่เขียนด้วยเทอร์โบปาสคาล(เมื่อ
พายล์แล้ว) จะยาวเกิน 64 กิโลไบท์
ด้ (ทำได้เหมือนกันแต่ยุ่งยากพอสมควร)

ขั้นตอนต่อไปเป็นหน้าที่ของโปรแกรม
ที่จะนำค่าทั้ง 2 นี้มาใช้ ในกรณีนี้คือ
oXY เมื่อจะส่งกลับ(return) เป็นหน้าที่
โปรแกรมย่อยที่จะยกเลิกค่าที่ผ่านมาทาง
ค (ไม่ว่าจะเป็นการผ่านค่าทางเดียวหรือ
ลับ เพราะไม่เกี่ยวข้องกัน ดังจะกล่าว
ต่อไป) การยกเลิกค่าที่ผ่านมาทางสแต็ค
หมายถึง โปรแกรมย่อยจะต้องถอนค่า return
address ในสแต็ค คืนให้แก่ IP ด้วย และจะ
ต้องเลื่อน SP กลับที่เดิมด้วย คือ ณ ตำแหน่ง
ที่เขียนไว้ในรูปที่ 1 ว่า "SP เดิม"

เมื่อใช้คำสั่ง RET เราจะได้ค่า return
address คืนให้แก่ IP ส่วนการเลื่อน SP นั้น
อินเทลได้ออกแบบ 8086 ไว้เพื่อการนี้โดย
เฉพาะคือ มีคำสั่ง RET n โดย n เป็นเลข
16 บิท ที่จะนำมาบวกเข้ากับ SP ตามตัวอย่าง
ที่ยกมาจะได้เป็น RET 4 (คือ RET 0004H)
เพราะเรามีข้อมูลที่จะยกเลิกอยู่ 4 ไบท์ (0014
H กับ 0028H) หลังจากปฏิบัติตามคำสั่งนี้แล้ว
จะทำให้ SP ไปชี้ที่เดิม

ค่าที่ผ่าน

เท่าที่กล่าวมาเป็นกรรมวิธีของการผ่าน
ค่าอย่างกว้างๆ ส่วนค่าที่จะผ่านแท้ๆ ยังขึ้น
อยู่กับรูปแบบของการผ่านค่าอีกคือ



รูปที่ 1 แสดงการเก็บค่าลงสแต็คเมื่อปฏิบัติตามคำสั่ง GotoXY (40, 20) ซึ่งจะเป็นการ CALL ไปยังโปรแกรมย่อย GotoXY รูป ก เขียนตามแบบที่นิยมเขียนหน่วยความจำที่กำหนดให้เป็นสแต็ค ซึ่งสำหรับ 8086 ในการเก็บค่าลงสแต็ค (push) มันจะลดค่า SP ลงหนึ่งแล้วเก็บค่า ณ ตำแหน่งที่ SP ชี้อยู่ ในการเก็บค่าจะเก็บครั้งละ 2 ไบท์ คือเก็บไบท์สูงที่แอดเดรสสูงและไบท์ต่ำที่แอดเดรสต่ำ (ลดค่า SP ลงอีกหนึ่ง) ส่วนรูป ข เขียนกลับกัน โดยเอาแอดเดรสต่ำไว้บนและแอดเดรสสูงไว้ล่าง ซึ่งจะสอดคล้องกับการกำหนดแบบของ struct ใน MASM ที่จะใช้ต่อไป

การผ่านค่าทางเดียว(Value Parameters) เป็นการผ่านค่าจากโปรแกรมผู้เรียกไปยังโปรแกรมย่อยเท่านั้น ไม่มีการส่งค่ากลับคืนมา (ดังตัวอย่างที่ยกมาเป็นการผ่านค่าทางเดียว) การผ่านค่าเช่นนี้ขึ้นอยู่กับแบบของข้อมูลอีกคือ

– ถ้าเป็นข้อมูลแบบ Integer, Byte, Boolean, Char และแบบที่ผู้ใช้กำหนดขึ้น จะผ่านทางสแต็คเป็นข้อมูลค่าละ 2 ไบท์ ถ้าข้อมูลใดมีไบท์เดียว เช่น Char จะทำให้ไบท์สูงเป็นศูนย์

– ข้อมูลแบบ Real ผ่านทางสแต็คเป็น 6 ไบท์

– ข้อมูลแบบ String จะผ่านทางสแต็คเป็นจำนวนไบท์เท่ากับขนาดของสตริงที่กำหนดบวกด้วยหนึ่ง เช่น String [40] จะผ่าน 41 ไบท์ ไบท์แรกจะอยู่ทางแอดเดรสต่ำ (top of stack) และไบท์แรกนี้จะเป็นจำนวนอักขระที่มีจริงในสตริงนั้น ตามด้วยอักขระตัวแรกและตัวต่อๆ ไป หากไม่เต็มสตริง ไบท์ที่เหลืออาจมีค่าใดก็ได้ (เทอร์โบปาสคาลได้กันเนื้อที่สำหรับสแต็คไว้ถึง 64 กิโลไบท์)

– ข้อมูลแบบ Sets ผ่านทางสแต็คเป็น 32 ไบท์

– ข้อมูลแบบ Pointers ผ่านทางสแต็คเป็นข้อมูล 4 ไบท์ คือเป็นเบส(base address) 2 ไบท์ และออฟเซท 2 ไบท์ (การเรียงลำดับคงเป็นแบบไบท์สูงอยู่แอดเดรสสูง และเบสอยู่ที่แอดเดรสสูงกว่าออฟเซท) ซึ่งเบสและออฟเซทนี้คือ แอดเดรสต้นของข้อมูลแบบพอยน์เตอร์ หากมีค่าเป็น nil ค่าทั้งหมดนี้จะเป็นศูนย์

– ข้อมูลแบบ Arrays และ Records จะผ่านค่าแอดเดรสไบท์ต้นของข้อมูลนั้นทางสแต็ค 4 ไบท์ เป็นเบสและออฟเซท

การผ่านค่าไปกลับ(Variable Parameters) เป็นการผ่านค่าจากโปรแกรมผู้เรียกและมีการส่งค่ากลับคืนมาจากโปรแกรมย่อย ที่กล่าวมานี้เป็นความรู้สึกของผู้เขียนโปรแกรม แต่ความเป็นจริงโปรแกรมย่อยจะไปดำเนินการโดยตรงกับข้อมูลในหน่วยความจำ การผ่านค่าแบบนี้ ไม่ว่าจะเป็นข้อมูลแบบใด

จะผ่านค่าแอดเดรสไบท์ต้นของข้อมูลนั้นทางสแต็ค 4 ไบท์ เป็นเบสและออฟเซท (ในบางแห่งจึงเรียกการผ่านค่าในแบบนี้ว่า pass by reference) ดังนั้นเมื่อโปรแกรมย่อยจะส่งกลับ(return) จะต้องยกเลิกค่าที่ผ่านมาทางสแต็คด้วย เพราะเป็นเพียงแอดเดรสซึ่งโปรแกรมที่เรียกมามีอยู่แล้ว และการเปลี่ยนแปลงค่าใด ๆ ได้ดำเนินการโดยตรงกับหน่วยความจำแล้ว จึงไม่มีการผ่านค่ากลับอย่างแท้จริง

ค่าของฟังก์ชัน หากโปรแกรมย่อยเป็นฟังก์ชัน คอมพายเลอร์จะสำรองเนื้อที่ในสแต็ค (เทียบได้กับการ push เป็นค่าแรก) เป็นค่าผลของฟังก์ชัน เป็นจำนวนไบท์ตามแบบข้อมูลของฟังก์ชันนั้น ตามที่กล่าวในหัวข้อ "การผ่านค่าทางเดียว" ดังแสดงในรูปที่ 2 แต่มีข้อที่ชวนฉงนคือ มีข้อกำหนดต่อไปอีกว่า ค่าฟังก์ชันของข้อมูลบางแบบให้ส่งกลับทางรีจิสเตอร์ ซึ่งถ้าเป็นกรณีนี้เราต้องยกเลิกสแต็คส่วนที่สำรองไว้เป็นค่าผลของฟังก์ชันด้วย สำหรับค่าที่ส่งกลับได้กำหนดไว้ตามแบบข้อมูลของฟังก์ชันดังนี้

– แบบ Integer, Byte, Char และที่ผู้ใช้กำหนดขึ้น ให้ส่งกลับทางรีจิสเตอร์ AX ถ้าเป็นข้อมูลไบท์เดียวต้องให้ AH เป็น 00H

– แบบ Boolean ส่งกลับทางแฟลก Z โดยถ้าค่าเป็นจริงให้เป็น NZ ถ้าเป็นเท็จให้เป็น Z

– แบบ Real ให้ส่งกลับทางสแต็ค (คือที่สำรองไว้) โดยให้ค่ายกกำลังอยู่ทางไบท์ต่ำ

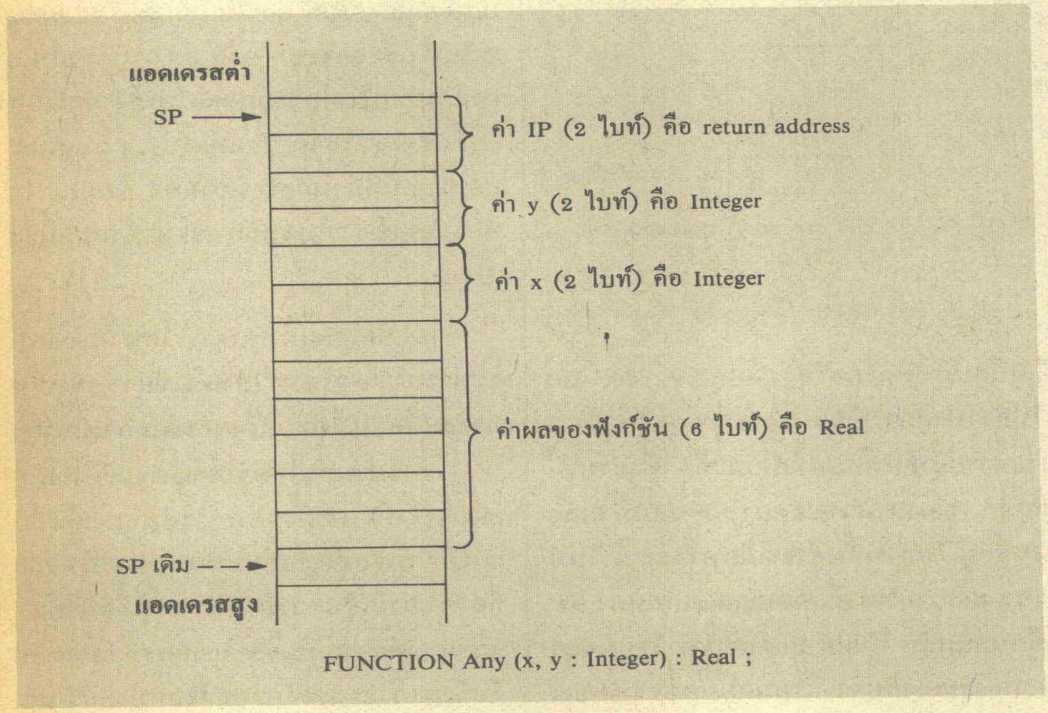
– แบบ set ให้ส่งกลับทางสแต็ค (คือที่สำรองไว้) โดยให้ SP ชี้ต้นสตริง (เซทเป็นข้อมูล 32 ไบท์ จึงถือเป็นสตริงด้วย)

– แบบอื่นที่ผ่านค่าโดยเบสกับออฟเซท ให้ส่งกลับทางรีจิสเตอร์ DX:AX ในแบบ เบส:ออฟเซท

กระดาษทด

เทอร์โบปาสคาลได้กำหนดไว้ว่า ห้ามเราใช้หน่วยความจำที่จัดไว้เป็นดาต้าเซกเมนท์ แต่ในการเขียนโปรแกรมเราอาจต้องการใช้หน่วยความจำ เพื่อเป็นพื้นที่ทำงานเปรียบเสมือนกระดาษทด จะทำได้โดยกันเนื้อที่ในสแต็คไว้ เช่น ต้องการเนื้อที่ 100 ไบท์ ก็เพียงแต่ใช้คำสั่ง

SUB SP, 100



รูปที่ 2 แสดงค่าในสแต็คเมื่อใช้ฟังก์ชัน Any

โดยที่เรารู้ค่า SP เดิม ซึ่งส่วนใ
จะถ่ายค่าให้แก่ BP (ดังจะกล่าวต่อไป)
สามารถที่จะใช้หน่วยความจำนี้ได้ อนึ่งกา
ห้ามไม่ให้เราใช้ดาต้าเซกเมนท์ ไม่ได้หม
ความว่าจะห้ามเราใช้รีจิสเตอร์ DS เรา
เก็บค่าเดิมไว้ในสแต็ค นำ DS มาใช้ และ
ค่าเดิมให้เมื่อเลิกใช้ เราสามารถสร้างด
เซกเมนท์ของเราเองได้ โดยให้ซ้อนกับส
แต่ใช้พื้นที่เฉพาะตรงที่เรากันไว้ หรื
ใช้หน่วยความจำของจอภาพซึ่งว่างอยู่

แบบของข้อมูลในภาษาเครื่อง

ในภาษาปาสคาลเราสามารถกำห
แบบของข้อมูลได้ตามที่เราต้องการ ในโ
แกรมแอสเซมเบลอร์รุ่นใหม่ เช่น MA
(Microsoft Assembler) ก็เปิดโอกาสใ
ทำได้เช่นกัน โดยเฉพาะที่ผมจะนำมาใ
บทความนี้คือ คำว่า STRUCT เป็นแอ
เซมเบลอร์ไดเรคตีฟ เทียบได้กับคำว่า
CORD ในปาสคาล และให้เรากำหนด
แบบขึ้นก่อน แล้วนำไปกำหนดตัวแปรเห
กับภาษาปาสคาลอีกเช่นกัน ดังที่แ
เปรียบเทียบในรูปที่ 3 (กรุณาอ่านคำอธิ
ใต้รูปด้วย) แต่ที่จะนำมาใช้นี้ จะนำแค่
ของมันมาใช้เท่านั้น เพื่อช่วยเราใน
อ้างแอดเดรสถึงข้อมูลในสแต็ค ดังจะแ
ในตัวอย่างต่อไป

เรียกใช้โปรแกรมย่อยภาษาเค
โดยตรง (External routine)

สิ่งที่อึดอัดใจผมคือตัวอย่างที่จ
มาแสดง ไม่ว่าจะคิดอะไรก็มีอยู่ในโพรซี
และฟังก์ชันมาตรฐานของเทอร์โบปา
เรื่องที่คิดขึ้นมาใหม่ก็ยาวเกินกว่าที่จ
มาเป็นตัวอย่าง ซึ่งเพียงเพื่อให้เห็นกร
ในการสร้าง และการใช้โปรแกรมย่อย
เครื่อง ผมจึงตัดสินใจที่จะยกตัวอย่างอั
ถึงแม้จะซ้ำกับของเดิม

ในภาษาปาสคาลแบ่งโปรแกรม
ออกเป็น โพรซีเดียวกับฟังก์ชัน ถึง

Pascal		MASM	
TYPE			
Invent	= RECORD	INVENT	STRUC
name	: String (30) ;	NAME	DB 31 DUP (?)
prize	: Real;	PRIZE	DB 6 DUB (?)
amount	: Integer ;	AMOUNT	DW
	END ;		ENDS
VAR			
Book : Invent		BOOK	INVENT
.....			
Book.amount : = 30 ;		MOV BOOK.AMOUNT, 30	
		{MOV[BX].AMOUNT, 30}	

รูปที่ 3 เปรียบเทียบการกำหนดตัวแปรแบบมีโครงสร้างในภาษาปาสคาลกับแอสเซมเบลอร์ของไมโครซอฟท์ (MASM) จะเห็นว่าขึ้นต้นด้วยการกำหนดแบบของข้อมูลเหมือนกัน การกำหนดแบบของข้อมูลนี้จะเป็นเพียงการกำหนดในคอมพายเลอร์หรือแอสเซมเบลอร์ ต่อมาเมื่อเรากำหนดตัวแปร book จะเป็นการกำหนดเนื้อที่ในหน่วยความจำจริง ๆ เพื่อใช้เก็บข้อมูลต่อไป และคำว่า book จะแทนแอดเดรสต้นของตัวแปรหรือก็คือข้อมูลนี้ ในโปรแกรมเราจึงสามารถอ้างถึงตัวแปรย่อยแต่ละตัวได้ดังที่เห็น โดยที่ชื่อตัวแปรย่อยแต่ละตัวจะเป็นออฟเซทไปจาก book

คำสั่งที่ยกมาเป็นตัวอย่างก็เพียงเพื่อแสดงให้เห็นว่า เราจะอ้างแอดเดรสถึงตัวแปรย่อยได้อย่างไร ส่วนที่เขียนในวงเล็บเป็นอีกแบบหนึ่งที่สามารถอ้างได้โดยให้เบสรีจิสเตอร์(BX หรือ BP) มีค่าเป็นค่าแอดเดรสของ book

ียนโปรแกรมย่อยเป็นภาษาเครื่องก็ต้องรูปแบบทั้งสองนี้ด้วย (แบบใดแบบ) การเขียนโปรแกรมย่อยในภาษาเครื่องมต้นด้วยการกำหนดส่วนหัว เช่นเดียวกับเขียนในภาษาปาสเช่นกัน โดยมีรูปแบบรูปที่ 4

ชื่อ คือชื่อ(identifier) ของโปรแกรมตามหลักการตั้งชื่อในภาษาปาสคาล การผ่านค่าผู้เขียนโปรแกรมย่อยจะต้องเป็นผู้กำหนดว่า จะให้ผ่านค่าใดมาบ้าง แบบใดผ่านทางเดียวหรือไปกลับ แบบของข้อมูลในโปรแกรมย่อยแบบ FUNCTION คือแบบข้อมูลของฟังก์ชันชื่อนี้ คำว่า EXTERNAL เป็นคำสงวน(reserved word) เพื่อแสดงว่าเป็นโปรแกรมย่อยภายนอก และอยู่ในไฟล์ตามชื่อที่ตามมา ไฟล์นี้เราต้องสร้างให้เป็นไฟล์ .COM หรือ .BIN เท่านั้น หาก ไม่กำหนดนามสกุล(.EXT) ว่าเป็น .BIN จะถือว่าเป็น .COM

ส่วนหัวนี้ เราต้องกำหนดไว้ในโปรแกรมภาษาปาสคาล เช่นเดียวกับโปรแกรมย่อยในภาษาปาสคาลเอง ต่างกันตรงที่มีเฉพาะส่วนหัวนี้เท่านั้น เมื่อคอมพายล์เครื่องจะโหลดโปรแกรมย่อยนี้โดยอัตโนมัติ และเป็นส่วนที่เทอร์โบปาสคาลเวอร์ชั่น 3.0 ได้พัฒนาไปจาก 2.0 ซึ่งใน 2.0 เราต้องโหลดโปรแกรมย่อยเองไปยังแอดเดรสที่กำหนดซึ่งยุ่งยากพอสมควร แต่ในเวอร์ชั่น 3.0 นี้ มีข้อจำกัดที่ต้องเขียนโปรแกรมย่อยในแบบรีโลเคต(relocatable) คือจะอยู่ในส่วนใดของหน่วยความจำก็ให้รันได้ (โดยการอ้างแอดเดรสสัมพันธ์) เพราะเราไม่ทราบล่วงหน้าว่ามันจะถูกโหลดลง ณ แอดเดรสใด

```
PROCEDURE ชื่อ(การผ่านค่า); EXTERNAL "ชื่อไฟล์.EXT";
FUNCTION ชื่อ(การผ่านค่า) : แบบข้อมูล; EXTERNAL "ชื่อไฟล์ .EXT";
```

รูปที่ 4 แสดงส่วนหัวของโปรแกรมย่อยภาษาเครื่อง

* สร้างฟังก์ชันภาษาเครื่อง

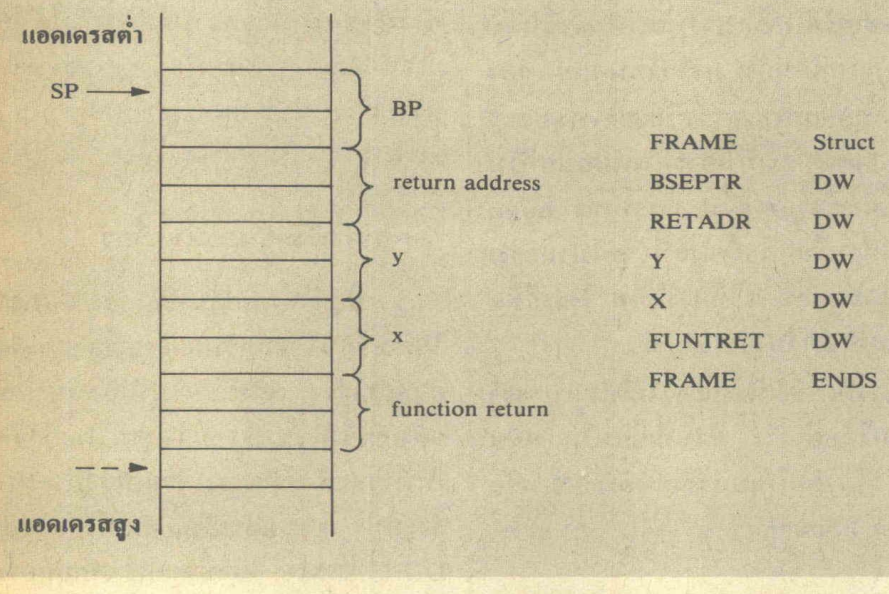
เมื่อกำหนดส่วนหัวแล้ว ขั้นตอนต่อไปคือเขียนโปรแกรมภาษาเครื่อง จะขอยกตัวอย่างว่า จะเขียนโปรแกรมหาผลต่างของเลขจำนวนจริง 2 จำนวน โดยให้ค่าคืนในรูปของฟังก์ชัน จะเขียนเป็นโปรแกรมได้ดังรูปที่ 5 และเก็บลงดิสค์โดยให้ชื่อว่า DIF1.ASM แต่เมื่อแอสเซมเบลอร์แล้วจะได้ชื่อไฟล์เป็นไฟล์ .COM ดังที่จะกล่าวต่อไป

ส่วนบนได้เขียนเป็นหมายเหตุแสดงให้เห็นว่า โปรแกรมนี้มีส่วนหัวในภาษาปาสคาลเป็นอย่างไร โปรแกรมเริ่มด้วยเก็บค่า BP(push bp) เพราะเราจะต้องใช้มันในการอ้างถึงข้อมูลในสแต็ค โดยที่ใน 8086 เมื่อเราอ้างถึง BP แอดเดรสจริงจะเป็น SS:BP โดยอัตโนมัติ และเป็นเรื่องที่กำหนดให้ใช้ การเก็บค่า BP จึงมีเป็นประจำทุกโปรแกรม

ก่อนที่จะอธิบายโปรแกรมต่อขอให้ดูค่าในสแต็คดังแสดงในรูปที่ 6 (ส่วนแบบของข้อมูลโครงสร้างทางขวาขอให้ผ่านไปก่อน) เนื่องจากเป็นฟังก์ชันแบบ Integer คอมพายเลอร์จะกันเนื้อที่ในสแต็คสำหรับ

```
; DIF1.ASM  TURBO Pascal external function
; FUNCTION Dif1(x,y : Integer): integer; EXTERNAL 'DIF1.COM'
;
cseg        segment
            assume  cs:cseg
SUB1        proc    near
            push    bp
            mov     bp,sp
            mov     ax,[bp + 6]
            sub     ax,[bp + 4]
            pop     bp
            ret     6
SUB1        endp
cseg        ends
            end
```

รูปที่ 5 โปรแกรมย่อยในภาษาเครื่องซึ่งจะเป็นฟังก์ชันในภาษาปาสคาล



รูปที่ 6 แสดงค่าในสแต็คเทียบกับการกำหนดแบบข้อมูล

ผลของฟังก์ชันไว้ 2 ไบท์ ดังที่กล่าวแล้วว่าเปรียบเสมือน push 2 ไบท์นี้ เป็นค่าแรก ค่าต่อไปจะเป็นค่า X, Y และ return address ค่าละ 2 ไบท์ ส่วนค่า BP อยู่ชั้นบนสุดของสแต็คเพราะเราพึ่งมา push ในโปรแกรมนี้

เมื่อเราถ่ายค่า SP ให้แก่ BP จะทำให้ BP ชี้ที่ชั้นบนสุดของสแต็ค คือตรงค่า BP เดิม ดังนั้นหากเราจะอ้างถึงค่า Y ก็จะเป็น BP+4 ถ้าเป็นค่า X ก็จะเป็น BP+6 เนื่องจากเราต้องการให้ dif1 = x−y จึงโหลดค่า X ให้แก่ AX ก่อนแล้วลบด้วยค่า Y ได้ในค่า AX ซึ่งตรงกับรีจิสเตอร์ที่กำหนดไว้ให้ส่งค่า Integer กลับพอดี

เมื่อจะกลับไปโปรแกรมที่เรียกมา เราต้อง pop ค่า BP คืน และต้องยกเลิกค่าในสแต็ค ซึ่งในกรณีนี้คือค่า x และ y รวม 4 ไบท์ กับเนื้อที่ที่กันไว้อีก 2 ไบท์ รวมเป็น 6 ไบท์ คำสั่งในการกลับของเราจึงเป็น RET 6

● ใช้ STRUCT ช่วย

โปรแกรมภาษาเครื่องนี้สามารถนำไปแอสเซมเบลอร์เพื่อใช้งานต่อไปได้ แต่กรรมวิธีอ้างถึงตัวแปรในสแต็คแบบนี้ อาจมีข้อผิดพลาดได้ เพราะเราต้องนับจำนวนไบท์ที่จะออฟเซทด้วยตัวเราเอง วิธีที่ดีกว่าคือนำการกำหนดตัวแปรแบบเป็นโครงสร้างมาช่วย ดังที่ได้แสดงไว้ในรูปที่ 6 ทางขวา

ซึ่งเมื่อเข้าใจแล้วเราไม่จำเป็นต้อง
รูปสแต็คอีกต่อไป ให้ดูตรงส่วนหัวประ
จะเริ่มโดยกำหนด

BSEPTR = 2 ไบท์ คือ DW

RETADR = 2 ไบท์ คือ DW

ซึ่ง 2 ฟิลด์นี้ จะเป็นเรื่องประจำ ต่อไ
ดูว่ามีการผ่านค่าตัวแปรใดบ้าง ไล่จาก
ขวามือมา ในกรณีนี้ ได้แก่ y และ x ซึ่ง
Integer และเป็นการผ่านค่าทางเดียว
เป็นการผ่านค่าข้อมูลโดยตรงจึงได้ว่า

y = 2 ไบท์ คือ DW

x = 2 ไบท์ คือ DW

ต่อไปให้ดูว่าเป็น PROCEDURE
FUNCTION ถ้าเป็นโพรซิเดียวก็จบ
กำหนดแบบของข้อมูล แต่กรณีนี้เป็นฟัง
ก็ต้องดูต่อไปว่าเป็นฟังก์ชันแบบใด ปร
ว่าเป็นแบบ Integer เราจึงเขียนต่อไปอี

FUNTRET = 2 ไบท์ คือ DW

ความจริงแล้วเราไม่ต้องกำหนด
FUNTRET ก็ได้ เพราะไม่ได้ใช้
กำหนดไว้เพื่อเตือนใจเรา เนื่องจากส่วน
จะนับไบท์หลัง RETADR มาเป็น
แก่คำสั่ง RET

นำไปเขียนเป็นโปรแกรมได้ตาม
7 จะเห็นว่าเราอ้างแอดเดรสถึงข้อมูล
สัมพันธ์กับ BP ได้ทันที และไม่ได้มี
กำหนดตัวแปร ทั้งนี้เพราะยืมเอาวิธีการ
เท่านั้น และถ้าสมมติว่าเป็นตัวแปร รีจิส
BP ก็ชี้ที่แอดเดรสต้นของตัวแปรอยู่แ

● การแอสเซมเบลอร์

ขั้นตอนต่อไป คือการแอสเซมเบ
ลิงค์ และเปลี่ยนไฟล์จากไฟล์ .EXE
.BIN หรือ .COM ขอให้ดำเนินการ
ลำดับดังนี้

```
> MASM DIF
> LINK DIF
> EXE2BIN DIF.EXE DIF.COM
```

ในแต่ละขั้นตอน เครื่องจะถามชื่อ
อีก ให้ตอบด้วยการกดคีย์ ENTER คื
ข้ามขั้นตอนนั้นไป เมื่อดำเนินการเสร็จ
จะได้ไฟล์ DIF.OBJ, DIF.EXE

```
; DIF.ASM  TURBO Pascal external function
; FUNCTION Dif(x,y : Integer): integer; EXTERNAL 'DIF.BIN'
;
FRAME     struc
BSEPTR    dw
RETADR    dw
Y         dw
X         dw
FUNCTRET  dw
FRAME     ends

CSEG      segment
          assume  cs:CSEG
SUB1      proc    near
          push    bp
          mov     bp,sp
          mov     ax,[bp].X
          sub     ax,[bp].Y
          pop     bp
          ret     6
SUB1      endp
CSEG      ends
          end
```

รูปที่ 7 โปรแกรมเดียวกับรูปที่ 5 แต่ใช้แบบของข้อมูลช่วยในการอ้างแอดเดรสถึงข้อมูลในสแต็ค

.COM ขอให้ลบไฟล์ DIF.EXE ไม่ให้สับสน อีกเรื่องหนึ่งคือไฟล์ DIF.M หากทิ้งไว้นาน ๆ อาจสับสน เพราะนด้วยตัวมันเองไม่ได้โดยสมบูรณ์

ผมเห็นว่าให้มีนามสกุลเป็น .BIN กว่า (ความจริงไฟล์ .BIN กับไฟล์ M ไม่ต่างกัน เพราะเพียงแต่ rename .BIN เป็นไฟล์ .COM ก็จะเป็นไฟล์ M โดยสมบูรณ์ ต่างกันตรงเราสั่งรันที่มีนามสกุล .BIN ไม่ได้) เมื่อต้องการนี้ คำสั่งสุดท้ายให้ใช้ว่า

EXE 2 BIN DIF

อนึ่งในการลิงค์จะมีข้อความบ่งข้อผิดพลาดว่า เราไม่ได้กำหนดสแต็คเซกเมนท์ ซึ่งเป็นเรื่องจริงและในกรณีนี้เราจะต้องไม่กำหนด จึงเพียงแต่รับทราบไว้เท่านั้น

● ทดลองใช้งาน

เมื่อเราสร้างโปรแกรมย่อยแล้วก็ควรทดลองใช้งาน ดังโปรแกรมในรูปที่ 8 ซึ่งให้ผู้ใช้ป้อนเลขจำนวนเต็ม 2 จำนวนเพื่อหาผลต่าง และจะพิมพ์ผลให้ทราบด้วย การทำงานให้อยู่ในวงรอบ UNTIL คือ

```
  type tstdif.pas
PROGRAM testDif(input,output);
VAR  x, y, z : Integer;
FUNCTION Dif(x, y : Integer):Integer; EXTERNAL 'DIF.BIN';

BEGIN
 WriteLn('Enter 2 number quit with 0');
 REPEAT
   Write('x = '); Read(x);
   IF x <> 0 THEN
     BEGIN
     Write('    y = '); Read(y);
     z := Dif(x,y);
     WriteLn('    ',x, ' - ', y, ' = ',z);
     END
 UNTIL x = 0;
END.
```

รูปที่ 8 โปรแกรมในภาษาปาสคาล ซึ่งเรียกใช้โปรแกรมย่อย Dif ในภาษาเครื่อง

เมื่อให้ป้อนเลขศูนย์จะออกจากโปรแกรมนี้ การคอมพายล์ทำได้ทั้งคอมพายล์ในหน่วยความจำเพื่อทดลองรัน หรือคอมพายล์เป็นไฟล์ .COM

*** โพรซิเดียวในภาษาเครื่อง**

การสร้างโพรซิเดียวในภาษาเครื่องง่ายกว่าการสร้างฟังก์ชัน ที่ผมนำมาต่อท้าย เพราะตัวอย่างที่จะยกมาแสดงค่อนข้างยาว สำหรับตัวอย่างผมได้แยกเป็น เมื่อเป็นการผ่านค่าทางเดียวตัวอย่างหนึ่ง และผ่านค่าไปกลับอีกตัวอย่างหนึ่ง แต่ในความเป็นจริงในโพรซิเดียวหรือฟังก์ชันหนึ่ง ๆ อาจประกอบด้วยการผ่านค่าหลาย ๆ แบบ

● โพรซิเดียวที่ผ่านค่าไปกลับ

ดังที่กล่าวแล้วว่าในการผ่านค่าไปกลับ โปรแกรมที่เรียกมาจะผ่านค่าแอดเดรส (4 ไบท์) ของข้อมูลมาทางสแต็ค โปรแกรมย่อยก็เพียงแต่นำค่าแอดเดรสนี้มาดำเนินการ โปรแกรมตามรูปที่ 9 จะเปลี่ยนอักษรภาษาอังกฤษในสตริง ซึ่งโปรแกรมผู้เรียกเป็นผู้ส่งมา จากตัวเล็กให้เป็นตัวใหญ่ อักขระนอกนั้นให้คงเดิม

โปรแกรมนี้ไม่ได้กำหนดแบบข้อมูล เนื่องจากมีข้อมูลเพียงหนึ่งเดียวคือแอดเดรสเท่านั้น เริ่มต้นโปรแกรมด้วยการให้ค่า SP แก่ BP ตามปกติ ที่ลบสแต็คด้วยค่า 20H เพื่อแสดงวิธีกันหน่วยความจำไว้เป็นกระดาษทด แต่ไม่ได้นำมาใช้งาน

คำสั่ง LES เป็นคำสั่งให้โหลดค่า 2 ไบท์แรกตามแอดเดรสที่อ้างถึงให้แก่รีจิสเตอร์ ซึ่งในกรณีนี้กำหนดให้เป็น DI และให้โหลด 2 ไบท์ถัดไป (ทางแอดเดรสสูง) ให้แก่ ES เมื่อปฏิบัติตามคำสั่งนี้แล้วจะได้ว่า ES:DI ชี้ไปที่ต้นสตริง ซึ่งเราทราบว่าเป็นจำนวนอักขระในสตริงที่มีอยู่จริง จึงโหลดมาให้แก่ CL เพื่อจะใช้ CX เป็นเคาน์เตอร์

เมื่อ INC DI จะทำให้ ES:DI ชี้ที่ข้อมูลไบท์แรก คำสั่งต่อไปเป็นการเทียบว่าเป็นอักขระตั้งแต่ a ถึง z หรือไม่ ถ้าใช่ให้ลบด้วย 20H จะได้เป็นอักษรใหญ่ และให้ทำงานเป็นวงรอบเท่ากับจำนวนอักขระ จะเห็นว่าโปรแกรมย่อยนี้ได้ดำเนินการเปลี่ยนแปลง

ข้อมูลในหน่วยความจำโดยตรง

หากดูในโปรแกรมภาษาปาสคาลคือ TESTUPR.PAS ในรูปที่ 10 จะเห็นว่าได้กำหนดตัวแปรขึ้นมาตัวหนึ่งคือ str และได้ให้ผู้ใช้ป้อนสตริง เมื่อผู้ใช้ป้อนเสร็จและกดคีย์ ENTER คำสั่ง ReadLn(str) จะนำค่าที่ผู้ใช้ป้อนไปเก็บในตัวแปรตัวนี้หรือคือหน่วยความจำหน่วยนี้ เมื่อเรียกใช้ UprCase ก็จะผ่านค่าแอดเดรสมาให้ และ UprCase จะเปลี่ยนค่าในหน่วยความจำดังกล่าวมาแล้ว เมื่อมาถึง WriteLn(str) ก็จะเป็นการพิมพ์อักษรใหญ่ทั้งหมด สำหรับการผ่านค่าไปให้แก่ WritrLn ซึ่งเป็นการผ่านค่าทางเดียว ทำได้อย่างไร จะแสดงให้ดูอย่างต่อไปนี้

```
D>type upr.asm
; PROCEDURE  UprCase(VAR Str : Strg); EXTERNAL 'UPR.BIN'
;            Strg = String[80]
;
CSEG        segment
            assume      cs:CSEG
UPER        proc        near
            push        bp
            mov         bp,sp
            sub         sp,20H
            push        es
            push        di
            push        cx
;
            les         di, dword ptr [bp + 4]
            mov         cl,ES:[di]
            mov         ch,00H
UP1:        inc         di
            cmp         ES:byte ptr [di],'a'
            jb          UP1
            cmp         ES: byte ptr [di], 'z'
            ja          UP1
            sub         ES: byte ptr [di], 20h
            loop        UP1
;
            pop         cx
            pop         di
            pop         es
            mov         sp,bp
            pop         bp
            ret         4
UPER        endp
CSEG        ends
            end
```

รูปที่ 9 โปรแกรมภาษาเครื่อง ซึ่งเป็นโพรซิเดียว UprCase ในเทอร์โบ-ปาสคาล

```
  type testupr.pas
{   TESTUPR.PAS      }

PROGRAM TestUper(input,output);
TYPE    Strg  = String[80];
VAR     Str : Strg;
PROCEDURE UprCase(VAR Str : Strg); EXTERNAL 'UPR.BIN';

BEGIN
 WriteLn ('Enter any String not more than 80 Char');
 REPEAT
   ReadLn(Str);
   UprCase(Str);
   WriteLn ('Here is the Upper case');
   WriteLn(Str); WriteLn;
   WriteLn ('Stop with enter quit');
 UNTIL Str = 'QUIT';
END.
```

รูปที่ 10 โปรแกรมภาษาปาสคาล ที่เรียกใช้โปรแกรมย่อย UprCase ในภาษาเครื่อง โดยถือว่าเป็นโพรซิเดียวของตน

● โพรซิเดียวที่ผ่านค่าทางเดียว

จุดหนึ่งที่ปาสคาลต่างกับซีคือ ผ่านค่าสตริง ในซีไม่ว่าจะเป็นการผ่านในแบบใดจะเป็นการผ่านแอดเดรส แต่สำปาสคาลหากเป็นการผ่านค่าทางเดียวจะหลัก pass by value คือผ่านค่าสตริงจริไปทางสแต็ค ผมจึงเลือกเรื่องของสตริเป็นตัวอย่าง

โปรแกรมย่อย PrintStr ตาม 11 จะนำสตริงไปแสดงทางจอภาพโดยคือไม่ผ่านระบบจัดการและให้แสดงattribute ที่ต้องการ คือให้เป็นภาพเน้นกะพริบหรือปกติได้ และจะให้เริ่ที่โรว์และคอลัมน์ใดก็ได้

ตัวอย่างนี้จะแสดงให้เห็นว่าการทำภาษาไทยให้ใช้กับโปรแกรมรูปทุกตัวจึงเป็นไปไม่ได้ เพราะการทำไทยให้เป็นมาตรฐานจะแก้ตรงระบการ หากไปเจอโปรแกรมที่ดำเนินการภาพโดยตรงเช่นโปรแกรมนี้ ก็จะใช้

ก่อนที่จะอธิบายโปรแกรมขอเรียทราบถึงเรื่อง attribute เสียก่อนคือ รุ่นเก่าเช่นเครื่องแอปเปิ้ลจะใช้ RAM 1 ไบท์ต่อภาพอักขระบนจอภาตัว แต่ในเครื่องไอบีเอ็มใช้ 2 ไบท์อักขระ โดยให้ไบท์หนึ่ง(ไบท์เลขคู่) ภาพอักขระเช่นเดิม ส่วนอีกไบท์หนึ่ง(เลขคี่) แสดง attribute คือทำให้เกิดเป็นกลับ(ขาว–ดำ) กะพริบและอื่น ๆ รวการให้ค่าสีต่าง ๆ ด้วย ได้แสดงattribute สำหรับภาพขาว–ดำไว้ใน 12

เมื่อเป็นเช่นนี้ ในโรว์หนึ่งซึ่งมีอักขระ จะต้องใช้หน่วยความจำ 160 แต่ในการอ้างโรว์จะอ้างจาก 0 ถึง 24 คอลัมน์จาก 0 ถึง 79 (สำหรับจอ

รูปที่ 11 โปรแกรมย่อยภาษาเครื่องแบบโพรซิเดียว แสดงถึงการผ่านค่าทางเดียว

```
pe prntype prnstr.asm

PROCEDURE PrintStr (str : Strg; row, col, att : Integer);
                        EXTERNAL 'PRNSTR.BIN'
   Strg = String[80]

AME     struc
PNTR    dw
ADR     dw
        dw
        dw
        dw
R       db
        db 80 dup (?)
ME      ends

G       segment
        assume  cs:CSEG
STR     proc    near
        push    bp
        mov     bp,sp
        push    es
        push    di
        push    si
        push    ax
        push    bx
        push    cx
        push    dx

        mov     ax,0B000H       ;screen base
        mov     es,ax

        mov     dx, [bp].COL    ;get row and col
        mov     ax, [bp].ROW
        xchg    al,ah
        mov     dh,ah
                                Change to offset
        mov     al, 0A0H        ;160 byte per line
        mul     dh              ;multiply by row
        mov     dh,0            ;
        add     dx,dx           ;
        add     ax,dx           ;plus col*2
        mov     di,ax           ;point to screen

        mov     ch,0            ;number of char
        mov     cl, [bp].CNTR   ;
        mov     ax, [bp].ATT    ;and attribute
        xchg    al,ah
        mov     si,0            ;index = 0
1:      mov     al, [bp + si].STR    ; WRITE TO SCREEN
        mov     es:[di], ax
        inc     si
        add     di,2
        loop    PRN1

        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     si
        pop     di
        pop     es
        pop     bp
        ret     87
NSTR    endp
EG      ends
        end
```

80×25) ดังจะเห็นที่ผมนำมาคำนวณหาค่าแอดเดรสตรงโรว์และคอลัมน์ที่ต้องการ อีกเรื่องหนึ่งคือหน่วยความจำที่จัดเป็น Vedio RAM ของจอภาพแบบโมโนโครมจะเริ่มจากแอดเดรสจริง E0000H ถึง E0FB9H

ขอให้สังเกตการกำหนดแบบข้อมูล ด้วยว่าหลังจาก return address แล้ว ได้กำหนดข้อมูลจากท้าย คือ att มาต้น คือ str แต่ได้แยกจำนวนอักขระในสตริงออกต่างหากเป็น CNTR ดังนั้นสตริงจริง ๆ คือ STR จึงมีเพียง 80 ไบท์ ในโปรแกรมได้ใช้ ES ร่วมกับ DI ในการชี้หน่วยความจำที่เป็น Vedio RAM โดยให้ ES มีค่าเป็น B000H ส่วนการหาค่าให้แก่ DI ทำโดยนำค่า row มาคูณด้วย 160 (จำนวนไบท์ต่อ row) แล้วนำไปบวกกับค่า col คูณ 2

จำนวนอักขระในสตริงได้อ่านมาให้แก่ CL และให้ค่า CH เป็นศูนย์ เพื่อที่จะใช้ CX เป็นตัวนับร่วมกับคำสั่ง LOOP ค่า attribute โหลดให้แก่ AH ส่วนการโหลดค่าอักขระในสตริงได้ใช้ SI เป็นอินเด็กซ์ร่วมกับ BP และค่า displacement คือ STR ตามตัวอย่างนี้ก็เหมือนกับเขียนว่า

MOV AL,SS: [BP + SI + 11]

ส่วนการนำไปเก็บใช้ ES ร่วมกับ DI โดยเก็บครั้งละ 2 ไบท์ โดยที่จะเก็บค่าใน AL ซึ่งเป็นไบท์ต่ำ และมีค่าอักขระ ในแอดเดรสเลขคู่ และค่าใน AH ซึ่งเป็นไบท์สูงและมีค่า attribute ในแอดเดรสเลขคี่ การเพิ่มค่า

FUNCTION	Attribute Byte							
	7	6	5	4	3	2	1	0
	FG	Background			Fouground			
Non Display	B	0	0	0	I	0	0	0
Underline	B	0	0	0	I	0	0	1
Normal	B	0	0	0	I	1	1	1
Reverse video	B	1	1	1	I	0	0	0

รูปที่ 12 แสดงภาพที่จะปรากฏบนจอภาพกับค่าที่ให้แก่ไบท์ที่เป็น Attribute
I คือ Intensity ให้เป็น 1 จะทำให้ภาพสว่างขึ้น
B คือ Blink ให้เป็น 1 จะทำให้ภาพกะพริบ

SI จึงเพิ่มหนึ่งส่วน DI ต้องเพิ่มสอง และจะทำงานเป็นวงรอบเท่าจำนวนอักขระในสตริงที่ส่งมา

จะเห็นว่าการนำภาพออกจอภาพโดยเนื้อแท้แล้วคือ การเคลื่อนย้ายข้อมูลจากหน่วยความจำที่เก็บข้อมูล ไปยังหน่วยความจำที่เป็น Vedio RAM แต่ในกรณีนี้ย้ายถึง 2 ครั้ง คือย้ายจากหน่วยความจำเดิมมาเก็บลงสแต็ค เพื่อผ่านค่าให้แก่ PrintStr และ PrintStr ย้ายไปยัง Vedio RAM (แต่ที่ส่วนใหญ่ต้องดำเนินการผ่านระบบจัดการ เพราะต้องดำเนินการในเรื่องอื่น ๆ ด้วยเช่น การขึ้นบรรทัด การเลื่อนหน้า ฯลฯ)

โปรแกรมเพื่อทดลองใช้โปรซิเดียวนี้ในภาษาปาสคาลคือ TestPrintString รูปที่ 13 ได้ทดลองใช้ค่า attribute ไว้เพียง 4 ค่า ให้พิมพ์ข้อความต่าง ๆ ดังที่เห็นค่าที่ผ่าน ได้ให้ไว้เป็นค่าคงตัว แต่จะให้เป็นตัวแปรก็ได้

● หากมีโปรแกรมย่อยจำนวนมาก

โพรซิเดียว PrintStr เป็นโพรซิเดียวหนึ่งที่น่าจะเก็บไว้ใช้งานต่อไป จากการดำเนินงานที่ผ่านมา จะเก็บอยู่ในไฟล์ชื่อ PRNSTR.BIN ส่วนไฟล์ PRNSTR.ASM จะเก็บไว้หรือไม่ก็ได้ แต่ส่วนมากจะเก็บไว้เพื่อการพัฒนาต่อไป แต่ถึงอย่างไรก็ตามควรจะทำคู่มือการใช้งาน อย่างน้อยที่สุดควรจดบันทึกไว้ ที่แนะนำให้ทำกันอีกเรื่องหนึ่งคือให้บันทึกไว้ในไฟล์ เช่น ไฟล์ READ.ME

หากคุณได้สร้างโปรแกรมย่อยไว้หลายโปรแกรม ไม่ว่าจะเป็นโพรซิเดียวหรือฟังก์ชัน หากเป็นเรื่องเดียวกันควรเก็บไว้ในไฟล์เดียวกัน ข้อดีคือลดจำนวนชื่อไฟล์ และไม่ทำให้ดูลานตาเมื่อไดเรคเตอรี่ดูชื่อไฟล์ แต่ข้อเสียคือ เมื่อใช้จะโหลดไฟล์นี้ทั้งไฟล์ลงหน่วยความจำ ไม่ว่าคุณจะใช้โปรแกรมย่อยในไฟล์นี้กี่โปรแกรม จึงควรเป็นเรื่องเดียวกันดังกล่าวแล้ว สมมติว่าคุณเขียนโปรแกรมย่อยเกี่ยวกับจอภาพต่อไปอีก 2 โปรแกรมคือ ClearScrren (ลบจอภาพ) SwapScr(สลับจอภาพ) ก็ควรจะเก็บไว้ในไฟล์ชื่อ SCREEN.BIN จะต้องเริ่มต้นด้วยการเขียนโปรแกรมให้กระโดดไปยังโปรแกรมย่อยที่ต้องการ เช่น

```
START JMP PRNSTR    ; PrintStr
      JMP CLRSCR    ; ClearScreen
      JMP SWAPSCR   ; SwapScr
```

สมมติว่าเป็นโพรซิเดียวทั้งหมด เราจะกำหนดโพรซิเดียวเหล่านี้ในส่วนข้อกำหนด (declrasation part) ได้ดังนี้

```
PROCEDURE PrintStr (...);
      EXTERNAL 'SCREEN.BIN';
PROCEDURE ClearScreen (...);
      EXTERNAL PrintStr (3);
PROCEDURE SwapScr (...);
      EXTERNAL PrintStr (6);
```

คอมพายเลอร์จะถือว่า เราได้ให้โปรแก
PrintStr อยู่ที่แอดเดรสแรก ส่วน
Screen อยู่ที่แอดเดรสแรก +3 และ S
อยู่ที่แอดเดรสแรก +6 แต่ตามคว
จริงเป็นแอดเดรสของคำสั่ง JMP
โปรแกรมที่ต้องการ (คำสั่ง JMP แบ
3 ไบท์)

ในวงเล็บที่ใส่จุดจุดจุดไว้ก็เ
การเขียนสั้น ๆ ที่จริงแล้วต้องเ
ผ่านค่าดังที่เคยกล่าวมา

ถ้อยแถลงภาษาเครื่อง (I
machine code)

เทอร์โบปาสคาลกำหนดให้
ถ้อยแถลง (statement) เป็นภาษา
โดยให้ขึ้นต้นด้วยคำว่า Inline ตาม
ภายในวงเล็บ มีรูปแบบเป็น

```
Inline (code/code { comment } /code/
```

ความหมายของคำว่า code
หมายถึงว่าอาจจะเป็น ตัวเลข ตัว
โพรซิเดียว ชื่อฟังก์ชัน หรือเป็น
นับในการกระโดด ซึ่งมีรายละเอีย
เราจะแอสเซมเบลอร์ด้วยมือ
ไปกล่าวในหัวข้อ "บนเครื่องแอปเ
comment หมายถึงคำอธิบายต้องอ
ก้ามปู โดยหลักใหญ่ก็เพื่อแสดงนิ
ของคำสั่งนั้น

คำว่าตัวเลขหมายถึงแมชีนโ
อาจจะเป็นไบท์เดียวหรือ 2 ไบท์ และ
ด้วยเลขฐานสิบ เช่น 250 หรือฐา
เช่น $5E หากเป็นเลขฐานสิบเมื่อ
255 เช่น 256 ก็จะเป็นเลข 2 ไบท์
ฐานสิบหกให้เขียน 4 หลัก เช่น $

การเขียนถ้อยแถลงภาษาเครื่
เป็นของ Z80 เช่น บนเครื่องแอปเ
สะดวกมาก เพราะเราสามารถแอสเซ
ด้วยมือได้ แต่บนเครื่องไอบีเอ็มซึ่ง
การแอสเซมเบลอร์ด้วยมือทำได้
ต้องใช้โปรแกรมแอสเซมเบลอร์เข้า

```
  type tstprn.pas
PROGRAM  TestPrintString(output);
CONST  norm = $07;
       rev  = $70;
       ind  = $0F;
       blnk = $87;
TYPE   Strg = String[80];
VAR    AnyStr : Strg;
PROCEDURE PrintStr (str : Strg; row, col, att : Integer);
                    EXTERNAL 'PRNSTR.BIN';

BEGIN
  ClrScr;
  GotoXY(1,20);
  Write('Pascal is ...');
  PrintStr('  This is the demo string  ',5,20, rev);
  PrintStr('This is the indent',7,25,ind);
  PrintStr('Here is the normal',9,25,norm);
  PrintStr('and this is the blinking one  ',11,20,blnk);
  Read;
  WriteLn('...here');
END.
```

รูปที่ 13 โปรแกรมทดลองใช้โพรซิเดียว PrintStr